```
function checkNNGradients(lambda)
%CHECKNNGRADIENTS Creates a small neural network to check the
%backpropagation gradients
%   CHECKNNGRADIENTS(lambda) Creates a small neural network to check the
%   backpropagation gradients, it will output the analytical gradients
%   produced by your backprop code and the numerical gradients (computed
%   using computeNumericalGradient). These two gradient computations should
%   result in very similar values.
%

if ~exist('lambda', 'var') || isempty(lambda)
    lambda = 0;
end

input_layer_size = 3;
hidden_layer_size = 5;
num_labels = 3;
m = 5;

% We generate some 'random' test data
Theta1 = debugInitializeWeights(hidden_layer_size, input_layer_size);
Theta2 = debugInitializeWeights(num_labels, hidden_layer_size);
% Reusing debugInitializeWeights to generate X
X  = debugInitializeWeights(m, input_layer_size - 1);
y  = 1 + mod(1:m, num_labels)';

% Unroll parameters
nn_params = [Theta1(:) ; Theta2(:)];

% Short hand for cost function
costFunc = @(p) nnCostFunction(p, input_layer_size, hidden_layer_size, ...
                               num_labels, X, y, lambda);

[cost, grad] = costFunc(nn_params);
numgrad = computeNumericalGradient(costFunc, nn_params);

% Visually examine the two gradient computations.  The two columns
% you get should be very similar.
disp([numgrad grad]);
fprintf(['The above two columns you get should be very similar.\n' ...
         '(Left-Your Numerical Gradient, Right-Analytical Gradient)\n\n']);

% Evaluate the norm of the difference between two solutions.
% If you have a correct implementation, and assuming you used EPSILON = 0.0001
% in computeNumericalGradient.m, then diff below should be less than 1e-9
diff = norm(numgrad-grad)/norm(numgrad+grad);

fprintf(['If your backpropagation implementation is correct, then \n' ...
         'the relative difference will be small (less than 1e-9). \n' ...
         '\nRelative Difference: %g\n'], diff);

end
```